2014 47th Annual IEEE/ACM International Symposium on Microarchitecture

PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration

Bo Su*, Junli Gu†, Li Shen*, Wei Huang†, Joseph L. Greathouse†, Zhiying Wang*

*State Key Laboratory of High Performance Computing
National University of Defense Technology
{subo, lishen, zywang}@nudt.edu.cn

†AMD Research
Advanced Micro Devices, Inc.
{Junli.Gu, WeiN.Huang, Joseph.Greathouse}@amd.com

***Abstract*— Performance, power, and energy (PPE) are critical aspects of modern computing. It is challenging to accurately predict, in real time, the effect of dynamic voltage and frequency scaling (DVFS) on PPE across a wide range of voltages and frequencies. This results in the use of reactive, iterative, and inefficient algorithms for dynamically finding good DVFS states. We propose PPEP, an online PPE prediction framework that proactively and rapidly searches the DVFS space. PPEP uses hardware events to implement both a cycles-per-instruction (CPI) model as well as a per-core power model in order to predict PPE across all DVFS states.**

We verify on modern AMD CPUs that the PPEP power model achieves an average error of 4.6% (2.8% standard deviation) on 152 benchmark combinations at 5 distinct voltage-frequency states. Predicting average chip power across different DVFS states achieves an average error of 4.2% with a 3.6% standard deviation. Further, we demonstrate the usage of PPEP by creating and evaluating a highly responsive power capping mechanism that can meet power targets in a single step. PPEP also provides insights for future development of DVFS technologies. For example, we find that it is important to carefully consider background workloads for DVFS policies and that enabling north bridge DVFS can offer up to 20% additional energy saving or a 1.4× performance improvement.

I. Introduction

Performance, power, and energy (PPE) are primary optimization targets in modern computers. For battery-powered systems such as tablets and phones, energy usage directly impacts usability, and power that is dissipated as heat must be removed without bulky heat sinks. Desktops and laptops have limited thermal headroom to share between integrated devices, often necessitating intelligent ways of balancing power usage [23]. Warehouse-scale systems must deal with varying electricity costs and power substation limitations [10]. CPUs are still the dominant component in these calculations, because they often dictate system performance and consume a majority of the total system power [2].

Dynamic voltage and frequency scaling (DVFS) is a widely used technique to optimize system energy efficiency [27], maintain power caps [5, 14, 18, 20, 21], and share power with other system components [23]. All of these involve periodically monitoring dynamic system behavior and adjusting the voltage and frequency (VF) state to meet specific power management targets.

Simple algorithms for controlling DVFS observe how the system is behaving during a time slice and will change VF states in reaction to these measurements. If the power goal is not met after this change, they will try a new VF state in the next time slice, iteratively approaching the goal. This mechanism is commonly used by commercial CPUs, but it can be inefficient, as a great deal of time may be wasted searching for the best solution.

Compared to such reactive algorithms, a predictive approach can be more efficient. For example, if an application is memory bound, a predictive algorithm may lower the CPU VF state just enough to achieve power savings without impacting performance. This requires good performance and power predictions in order to make the right decision in a single step. Unfortunately, existing predictors have limitations.

It can be difficult to accurately predict performance across frequencies due to interactions with devices (such as DRAM) in other clock domains. There have been many proposed hardware additions that enable such estimates [8, 17, 22, 25], but they are generally unavailable in commercial processors. We recently described a method for accurately estimating one of these mechanisms, leading load counters, using existing performance monitoring hardware [28], though we did not demonstrate it in a proactive DVFS manager.

More problematically, it is difficult to estimate power at the current VF state and to predict power usage at other VF states [20]. Off-chip power monitors may measure the whole chip, making it difficult to separate the effects of individual cores. While core-event-based estimation has been shown in simulation [24], demonstrations on real hardware have been less accurate [27]. This leads to a need for specialized hardware [13], per-application offline profiling [14], or external power monitors [20] in order to know a core's current power usage. Previous work on estimating power at other VF states has used simple models that do not take into account workload variation, such as assuming a direct relationship between total power usage and either frequency [29, 14] or the number of executed instructions [27].

This paper introduces PPEP, an online PPE prediction framework that uses execution statistics gathered on real processors to estimate PPE at the current VF state and predict

1072-4451/14 $31.00 © 2014 IEEE
DOI 10.1109/MICRO.2014.17

PPE at all other states. PPEP is a software-level tool that does not require per-application offline training, external power monitors, or new hardware. PPEP periodically reads hardware performance counters from the CPU cores, allowing it to quickly adjust to program phase changes. It also models the power of components outside the core such as the north bridge (NB). It then predicts how the program would run at different VF states, allowing accurate PPE decisions to be made in a single step. While we focus our study on the CPU in order to better characterize our prediction mechanism, PPEP could also be included in system-level models, such as CoScale [6], as a more accurate CPU estimator.

This paper makes the following contributions:

- We develop an online PPE prediction model, PPEP, which uses hardware performance events and on-chip temperature measurements to achieve an average chip power estimation error of 4.6% versus a commercial AMD CPU. When used to predict full-chip average power across vastly different VF states, PPEP shows an average error of 4.2%.
- We demonstrate that a predictive DVFS controller can use PPEP to explore the energy-delay space and pick energy- and EDP-optimal points with high accuracy.
- We show a power capping mechanism built with PPEP that operates 14× faster than a common reactive model.
- We use PPEP to explore future DVFS designs and estimate that an NB with multiple VF states could achieve up to 20.4% additional power saving or 1.37× higher performance.

This paper is organized as follows. Section II describes the hardware and benchmarks that were used in this study. Section III describes our cycles-per-instruction (CPI) predictor, and Section IV details our model of per-core power consumption that works across VF states. Section V uses these models to explore the ED space of programs and provides insights for future DVFS technology. Section VI is the related work, and the paper concludes with Section VII.

II. Experimental Methodology

We use real-time hardware measurements to derive our performance and power prediction models. To measure CPU power data, we use a Pololu ACS711 Hall effect current sensor clamped onto the +12V ATX power line of the CPU, between the power supply and the voltage regulator on the motherboard. An Arduino board with an AVR microcontroller is used to sample the output of the sensor, generate the power readings and send them to the processor via a USB-based serial port. This setup is similar to those used in previous work [4, 7]. We take a power reading through this setup every 20ms and record these to a power trace. We use 10 power readings for every DVFS decision interval, making each interval 200ms.

The systems under test ran Ubuntu 12.04 LTS using kernel version 3.2.0-24. For temperature measurements, we use the socket thermal diode readings accessible through the *hwmon* tree in *sysfs*. When gathering online performance statistics, we use *taskset* to affinitize benchmarks to particular cores and *msr-tools* to set and read performance counters.

Our main experimental platform is an 8-core AMD FX-8320 processor running on an ASUS M5A97 R2.0 motherboard with two 4GB DDR3 DRAM DIMMs. The AMD FX-8320 processor has four compute units (CU), each containing two CPU cores and 2MB of shared L2 cache. All of these cores share an NB that includes the memory controller and 8MB of L3 cache. Each CU supports five software-visible VF states. The five VF states of our processor are VF5 (1.320V, 3.5GHz), VF4 (1.242V, 2.9GHz), VF3 (1.128V, 2.3GHz), VF2 (1.008V, 1.7GHz), and VF1 (0.888V, 1.4GHz). The cores also have two hardware-controlled boost states that are enabled only in VF5. We disabled these because they are not software controllable; unexpectedly entering a boost state would affect the power and event counts that we measure. Our PPEP technique could be used by hardware or firmware boost controllers, but we disable boosting to maintain control of our experiments.

The AMD FX-8320 processor can also perform per-CU power gating (PG). We first build a power model when PG is disabled in the BIOS (Section IV-A to IV-C). After that, we analyze the power difference when PG is enabled (Section IV-D) and derive a new power model.

We also verify the accuracy of our model on an AMD Phenom™ II X6 1090T processor, which supports four VF states but does not support power gating. The results of our study of the AMD FX-8320 are shown in the figures, while the results from the AMD Phenom II processor are described only in the text due to space constraints.

We test our models on 3 benchmark suites, which we combine to make 152 benchmark combinations: SPEC® CPU 2006 v1.2 [12], PARSEC v2.1 [3], and the NAS Parallel Benchmarks (NPB) v3.3.1 [1]. We consider multi-threaded runs for PARSEC and NPB, and multi-programmed runs for SPEC CPU2006. We split our benchmarks into four equal sets and test our models using cross validation.

III. Performance Prediction Model

The first step in the process of building a power and energy manager is to build an accurate performance prediction model. To achieve this, we implement a mechanism that can predict the cycles-per-instruction (CPI) at one VF state from measurements taken at another VF state. We recently described an approximation of a *leading loads* predictor [8, 17, 25] that is built using miss address buffer (MAB, commonly known as a miss status handling register) occupancy information on AMD CPUs. We build on this LL-MAB performance predictor in this work [28].

Leading loads performance predictors split the execution of a program into *core time*, which scales as frequency changes, and *memory time*, which stays constant. A leading

load is the first demand miss to leave a core's clock domain whenever no other leading load from that core exists. This, in effect, avoids measuring the latency of loads caused by memory-level parallelism (MLP) while still taking cache effects and variable latencies into account. Miftakhutdinov et al. demonstrated that this model is not ideal (especially when bandwidth is constrained) [22], but this model has the benefit of existing in contemporary hardware. In AMD CPUs, performance monitors can measure the amount of time that an off-core memory access is in the highest priority MAB, which is a good approximation of a leading load.

We previously demonstrated that this yields accurate performance predictions when estimating the whole-program execution time of single-threaded applications. In this work, we improve upon the existing LL-MAB model by extending it to build an online CPI predictor which operates on periodic execution intervals. We divide CPI into core CPI (CCPI) and memory CPI (MCPI). CCPI stays the same across VF states while MCPI scales proportionally with frequency. In each prediction interval, we gather the CPI and MCPI in order to calculate CCPI. This requires three performance counters to implement. CPI is calculated by *CPU_Clocks_not_Halted* divided by *Retired_Instructions*. Similar to our LL-MAB model, MCPI is calculated by *MAB_Wait_Cycles* divided by *Retired_Instructions*. Thus, when running at frequency *f*, one can predict the CPI at frequency *f'* as shown in Equation 1.

$$CPI(f') = CCPI(f) + MCPI(f) \times \frac{f'}{f} \qquad (1)$$

Because this model predicts periodic CPI rather than whole-program execution time (which our previous work demonstrated), we tested its prediction accuracy on single-threaded versions of the 52 benchmarks from our three test suites. We measured the above-mentioned performance counters every 200ms at both VF5 and VF2. A simple comparison of the predicted and measured CPI values from each sample would not properly test the model, as the execution time of the program is different at the two different frequencies.

Instead, after gathering the performance counter traces, we divide them into segments based on the number of instructions completed. We then sum the number of cycles we predicted that segment to take (based on the performance counters in the other trace) and compare to the number of cycles that segment actually took. The difference between these two is the prediction error.

Our LL-MAB CPI predictor had an average error of 3.4% when predicting from VF5 down to VF2 and 3.0% when predicting from VF2 up to VF5. The standard deviations were 4.6% and 3.2% respectively. These values are similar to those measured in our previous work, so we believe this to be a good CPI predictor.

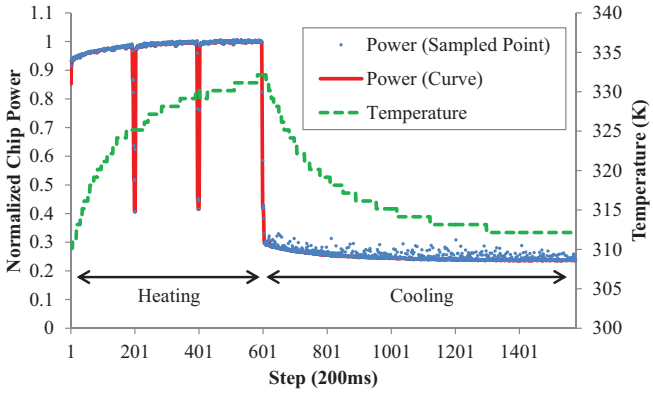


Figure 1 Idle power and temperature at VF5 as the workload changes on an AMD FX-8320.

IV. Per-core Power Prediction Model

In this section, we present a temperature-aware per-core idle power model and a dynamic power model based on performance monitor counters (PMCs). These can provide per-core online power estimates without requiring application profiling beforehand. Through measurements and microarchitectural analysis, we derive a method for scaling dynamic power based on hardware event counts taken at various VF states. In addition, we extend our power estimation model to predict power consumption at other VF states.

A. Chip Idle Power Model

We define chip idle power to include both static leakage power and active (not power gated) idle dynamic power when only OS housekeeping threads are running. We use an experiment, shown in Figure 1, to examine the relationship between idle power, VF states, and temperature. First, we run heavy workloads to heat up the processor until it reaches a steady-state temperature. We then stop the work. The chip remains active (not power gated) but idle at the VF state of our choice while it cools. During the cooling process, power and temperature traces are collected. These include leakage power as a function of temperature and a constant dynamic power, due to OS housekeeping threads, that exists whenever the chip is idle but not power-gated.

Theoretically, the relationship between leakage power and temperature should be exponential. However, we observed that, when the variation ranges of power and temperature are small, the relationship of idle power and temperature is close to linear. A linear relationship significantly simplifies the model training process and is a reasonable approach if we are mostly interested in temperatures within the normal operating range of our processor. Similar linear relationships have been previously observed [19].

We use this data to build a regression model for the chip idle power based on temperature and voltage, as shown in Equation 2. T is the temperature in kelvin, and both $W_{idle1}(V)$ and $W_{idle0}(V)$ are third-order polynomial

Table I Selected hardware events on an AMD FX-8320 processor. (E1-E9 for dynamic power model; E10-E12 for performance model)

NO.	Event Code	Event Name
E1	PMCx0c1	Retired_UOP
E2	PMCx000	FPU_Pipe_Assignment
E3	PMCx080	Instruction_Cache_Fetches
E4	PMCx040	Data_Cache_Accesses
E5	PMCx07d	Request_To_L2_Cache
E6	PMCx0c2	Retired_Branch_Instructions
E7	PMCx0c3	Retired_Mispredicted_Branch_Instructions
E8	PMCx07e	L2_Cache_Misses
E9	PMCx0d1	Dispatch_Stalls
E10	PMCx076	CPU_Clocks_not_Halted
E11	PMCx0c0	Retired_Instructions
E12	PMCx069	MAB_Wait_Cycles

functions of voltage. We believe this is reasonable because the leakage part of idle power is mostly exponentially related to voltage in sub-100nm technologies. In addition, the active idle power, frequency has a linear relationship to voltage. Therefore, a third-order polynomial function can accurately capture both leakage and idle active power in a simplified model. Having a unified idle power model avoids storing a static power table, as required by existing work [27]. Ambient temperature's impact on leakage is also naturally included in the temperature term. All cores share the same voltage rail in the processors used for this study. However, our methodology can be extended to future processors with per-core voltage rails. We verify the chip idle power model on an AMD FX-8320 processor across each VF state. The average absolute error (AAE) for VF5 down to VF1 is 2%, 3%, 4%, 3%, and 3%, respectively. On the AMD Phenom™ II X6 1090T processor, the AAE for VF4 to VF1 is 3%, 2%, 2%, and 2%, respectively.

$$P_{idle}(V) = W_{idle1}(V) \times T + W_{idle0}(V) \tag{2}$$

B. Chip Dynamic Power Model

In order to build an estimate of dynamic power usage, we use a regression model that takes nine power-hungry hardware event counts as inputs. The model is built on information collected at VF5 and then scaled to other VF states, which is a one-time, offline effort. After that, it gives estimates of dynamic power by reading performance counters, without relying on power meters and current sensors.

1) Power Model Construction: A processor's dynamic power depends on circuit switching activity in its core and the north bridge (NB). Among all collectible activities, we identify nine hardware events, shown in Table I, which are highly correlated to dynamic power. The first seven represent the core activity, including the pipeline, and both the L1 and L2 caches.

We attribute part of the NB's power to each core, since they share it. To do this, we can only choose NB-related PMCs that can be collected on a per-core basis, rather than any events counted in the shared NB. We use *L2_Cache_Misses* and *Dispatch_Stalls* for this. The former represents L3 cache access operations from the core measuring the events. The latter is usually caused by load/store queues or OoO storage (e.g., reservation stations) being full. Load/store queue stalls are usually due to the long latency of the last-level cache or off-chip memory accesses, which happen in the NB. Therefore, we found that *Dispatch_Stalls* can help approximate the NB activity caused by a core.

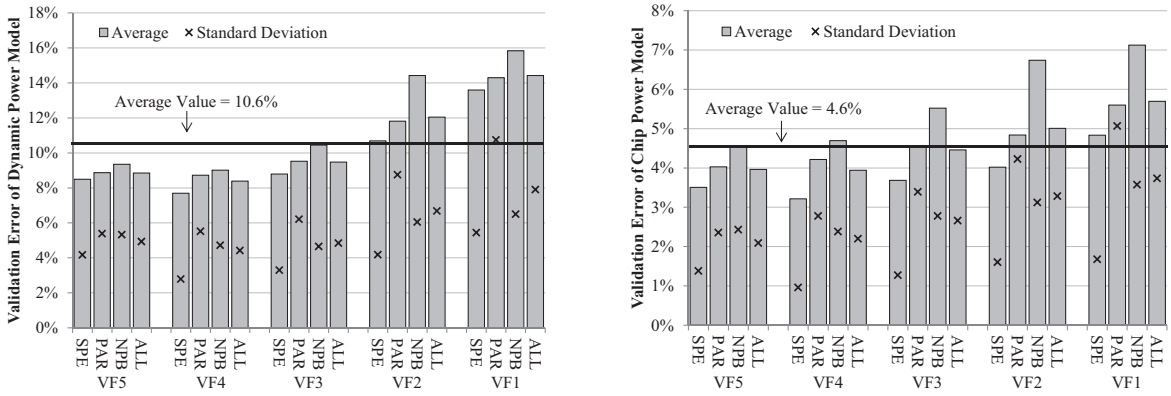
We build our dynamic power model based on a linear regression that uses the events E1-E9 shown in Table I. We use the three additional events, E10-E12, for the performance predicting model described in Section III. We run a total of 152 benchmarks taken from SPEC® CPU2006, PARSEC, and NPB to get the performance counter, power, and temperature traces at a 200ms sampling interval. For SPEC CPU2006, we run 61 multi-programmed benchmark combinations: 29 single-programmed, 15 double-programmed, 10 triple-programmed, and 7 quad-programmed. For PARSEC, we use 51 multi-threaded runs of the Pthreads (OpenMP for freqmine) version using "native" input set. For NPB, we use 40 multi-threaded runs of the OpenMP versions. We divide the 152 benchmarks evenly into four groups and use three of them to perform any particular training run.

In each 200ms interval, we measure the chip power every 20ms and use the average value as the power of that interval. Dynamic power is calculated by subtracting the calculated idle power from the measured power. The AMD FX-8320 processor contains six performance counters for each core, and we time-multiplex them in order to measure all 12 events needed for both the CPI predictor and the power model.

$$P_{dyn} = \sum_{c=0}^{7}(\sum_{i=1}^{7}((\frac{V_n}{V_5})^{\alpha} \times W_{dyn(i)} \times E_i) + \sum_{i=8}^{9}(W_{dyn(i)} \times E_i)) \tag{3}$$

Our dynamic power model is shown in Equation 3. We use i, c, and n to represent the event ID, core ID, and current VF state. E_i is the per-second counts for event *i*. By adding counts of the same event in different cores, we get a nine element vector for each 200ms interval. The coefficients, $W_{dyn(i)}$, are derived from linear regression using the data from VF5. To generalize the model, we use voltage to scale $W_{dyn(1)} - W_{dyn(7)}$ when the cores run at different VF states. The exponential factor α for voltage scaling is a constant that we derived from actual measured power at different voltages and is unique for each process technology. For the two NB events, we choose to use the system default setting and keep the NB VF state constant in all of our tests.

2) Power Model Validation: To validate the accuracy of the power model, we randomly split our collection of 152 benchmarks into four equally sized sets and perform 4-fold



(a) Validation error of the dynamic power model.

(b) Validation error of the chip power model.

Figure 2 Validation error of the dynamic power model (a) and the chip power model (b) at each VF state. The bars are the averages of AAEs. The crosses are the standard deviations of AAEs.

cross validation. This method uses all combinations of three sets for training and tests the accuracy of our model on the remaining set for all of the training collections. This ensures that we test against all of our benchmarks without testing on our training data at any point. We measure the absolute error of the estimated (versus measured) power at each interval for a given benchmark and VF state. We then gather the AAE across all 200ms samples.

Figure 2(a) shows the error of the *dynamic power model* on our AMD FX-8320. Due to space limitations, we only present results at the granularity of each benchmark suite. The error information includes the average value of the AAEs as well as the standard deviation of each benchmark suite.

The AAE of the dynamic power estimates for all benchmark combinations at all VF states is 10.6%. The AAEs across VF5 to VF1 are 8.9%, 8.4%, 9.5%, 12.0%, and 14.4%, respectively. VF1 shows larger errors because: (1) lower VF states have relatively smaller absolute power values, and hence the larger *percentage* error; and (2) the coefficients are derived at VF5, so the further VF states show a relatively higher error. We also see that the average standard deviation (SD), as shown by the cross, is 5.8% over all suites. We do see a few outliers, with a maximum error up to 49%. This happens in *DC* and *IS* from NPB, and *dedup* from PARSEC. Possible sources of error are: (1) these benchmarks have rapid phase changes, which may cause errors because of our performance counter multiplexing; and (2) *dedup* and *IS* have much shorter execution times than the other benchmarks and may be poorly represented by the training data.

Combining the idle and dynamic power models yields the *full-chip power model*. Figure 2(b) shows the accuracy of this model. For all benchmark combinations at all VF states, it achieves a 4.6% average AAE with an average SD of 2.8%.

We also validate the model on our AMD Phenom II processor using PARSEC and NPB from VF4 to VF2. The dynamic power estimates over these points have an average AAE of 8.2%, 7.3%, and 7.1%, respectively, while the full-chip model has average AAE of 3.6%, 3.1%, and 2.6%.

C. Predicting Power across VF States

The above model works by gathering performance counters at the VF state to be estimated. It would be helpful to predict power at other VF states during runtime without actually switching to that state and gathering performance counters. Unfortunately, simple linear scaling of power based on frequency and voltage does not take into account the possible change in workload characteristics. For example, a workload can become memory bound at higher frequencies, meaning some counters will scale in non-linear ways.

1) Hardware Event Prediction: The key challenge in making this estimate is predicting the hardware event counts at other frequencies. Through experimentation, we made two observations. Combining them with our performance prediction model, we are able to predict hardware event counts, and hence power consumption, at different VF states.

- *Observation 1: At any given point in the execution of a program, core-private hardware event counts per instruction are independent of VF state.*

Observation 1 indicates that the activity of a core's private resources incurred per instruction only depends on the core microarchitecture and the application's behavior. This is like the fingerprint of the microarchitecture and application. The core's private resources include the pipeline, the L1 caches and accesses to the L2 cache. Our experiments show that the first eight hardware events for the dynamic power model happen in these core's private resources and all follow Observation 1 very well. We use the same configuration and benchmarks we used in Section III to verify this observation. For E1 to E8, the difference between VF5 and VF2 on an AMD FX-8320 processor is 0.6%, 0.9%, 0.7%, 5.0%, 0.7%, 1.3%, and 4.0%, respectively. We observe similar results on the AMD Phenom II processor.

- *Observation 2: At any given point in the execution of a program, (*CPI - Dispatch_Stalls_per_inst*) is independent of VF state.*

Observation 2 indicates that the dynamic gap between *CPI* and *Dispatch_Stalls_per_inst* changes along with program phase, but stays the same across all VF states at any given point in the execution of a program. We use the same configurations and benchmarks that we used in Section III to verify this observation. The difference in the gap across VF5 and VF2 on an AMD FX-8320 processor is 1.7%. We observe similar results on the AMD Phenom™ II processor. This observation is critical for predicting power across VF states. At any particular point in a program, we can use the predicted *CPI* from our performance prediction model to predict *Dispatch_Stalls_per_inst* using simple arithmetic.

This phenomenon can be explained as follows. We can divide a core's unhalted clock cycles into two parts: retiring cycles and bubble cycles.

Retiring cycles are cycles with architectural states commitments from instructions. On an ideal pipeline, the number is equal to the commit width, which would be roughly equal to the issue width in a balanced design. Applications do not always retire *Issue_Width* instructions in every retire cycle, but interval analysis uses this simplified assumption and shows good analytical modeling results [9]. We therefore estimate that a retire cycle is a cycle with *Issue_Width* instructions retiring. We later discuss how this could be generalized.

Bubble cycles are those cycles where no instructions retire. These can further be divided into *stall cycles* and *discarded cycles*. Stall cycles are cycles where the core is stalled waiting for data from the memory hierarchy or due to pipeline resource limitations such as reorder buffer occupancy. Discarded cycles are those where no useful work is accomplished due to pipeline flush events such as branch mispredictions and interrupts. Equation 4 shows the relationship between unhalted, retiring, stall, and discarded cycles.

$$\begin{aligned} C_{unhalted} &= C_{retiring} + C_{bubble} \\ &= C_{retiring} + C_{stall} + C_{discarded} \end{aligned} \quad (4)$$

$$E10 \approx E9 + E7 \times MisBranchPen + \frac{E11}{Issue\_Width} \quad (5)$$

$$\begin{aligned} CPI - DispatchStall\_Per\_Ins \approx \frac{1}{Issue\_Width} \\ + MisBranchPen \times \frac{Inst\_MisPred}{Inst\_Retired} \end{aligned} \quad (6)$$

The unhalted cycles can be counted by event E10, *CPU_Clocks_not_Halted*. The stall cycles can be counted by event E9, *Dispatch_Stalls*. The ideal retiring cycles can be calculated by the ratio between event E11, *Retired_Instructions*, and issue width (which is a microarchitectural feature). Due to hardware limitations, we use mispredicted cycles to approximate discarded cycles. Mispredicted cycles can be expressed as the product of event E7, *Retired_Mispredicted_Branch_Instructions*, and the resulting penalty cycles (*MisBranchPen*). We find that this approximation does not significantly increase power prediction error due to its relatively small weight in Equation 3. Therefore, Equation 4 can be rewritten in terms of event counts as Equation 5. Dividing both sides of this equation by the retired instructions and rearranging the terms yields Equation 6.

Among terms in the right-hand side of Equation 6, *Issue_Width* and *MisBranchPen* are microarchitecture-specific constants, and *Inst_Retired* and *Inst_MisPred* are constants at a given program point. Since none of these values are dependent on frequency, *CPI - DispatchStall_Per_Inst* is approximately a constant at a given point in a program, regardless of VF states.

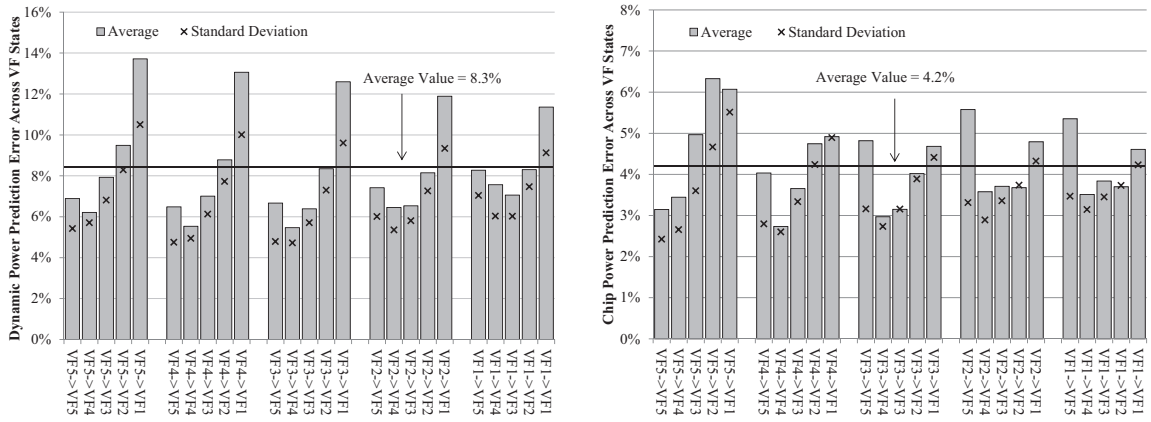
We note here that Equation 5 assumes that every retiring cycle retires *Issue_Width* instructions. We could make a more accurate estimate of this value by separating out *Cycles_Retiring_1*, ..., *Cycles_Retiring_Issue_Width*. Unit mask values in our performance counters could count these values separately, at the cost of more counter multiplexing. We note that this retire count is mostly a function of program location and microarchitectural features, however. As such, Observation 2 still holds.

The above two observations provide the relationship of hardware event counters among different VF states. Therefore, given the hardware event counts at one VF state, we can derive good estimates of event counts at any other VF state.

2) Validation of Power Prediction across VF States: We now evaluate the power prediction errors for all the benchmark combinations across all pairs of VF states. For each pair of VF states, we compare the average measured power and the average predicted power for each benchmark combination and illustrate the error for each suite.

Figure 3 shows the validation results of dynamic power and chip power with predictions across different VF states. VF*i*→VF*j* means predicting power at VF*j* based on event counts gathered at VF*i*. We use the same 4-fold cross validation method as was used for the power model, and for each VF state pair, the prediction errors of all 152 benchmark combinations are collected and presented in one column.

As Figure 3(a) shows, the average prediction error for the dynamic power ranges from 5.5% to 13.7% in different VF state pairs, with an overall average error of 8.3%. As the voltage and frequency differences between VF*i* and VF*j* increase, the prediction errors usually increase accordingly. The errors are higher when the target state is VF1 because the dynamic power model has higher error there. Figure 3(a) also presents the standard deviation, which ranges from 4.7% to 10.5%, with an overall average of 6.9%.



(a) Prediction error of the dynamic power across VF states

(b) Prediction error of the chip power across VF states

Figure 3 Dynamic (a) and chip power (b) prediction error across VF states. The bars are the averages of AAEs. The crosses are the standard deviations of AAEs.

Figure 3(b) shows the across-VF errors for the chip level power prediction. The average prediction errors for each VF state pair range from 2.7% to 6.3%, with an overall average error of 4.2%. The standard deviations for all VF state pairs fall within 5.5%, with the average deviation as 3.6%.

On the AMD Phenom II X6 1090T processor, we verify the prediction accuracy between VF4, VF3, and VF2. The overall average prediction error for dynamic power model and chip power model are 5.6% and 3.1%, respectively.

D. Power Gating and Per-core Idle Power

In our AMD FX-8320 processor, power gating (PG) is implemented at the CU level. That is, when PG is enabled in the BIOS, a CU can be power gated if both of its cores are idle. The NB can also be power gated if all of the CUs are idle. To model the impact of PG on chip idle power and to isolate the core idle power from NB idle power, we developed a microbenchmark, *bench_A*, which has an L1-resident data set, requires no dynamic NB accesses, and has a steady program phase. The performance and dynamic power of each instance is the same if multiple instances are running concurrently on different CUs.

We devised an experiment in which we fix the processor's VF state and sweep the number of busy CUs from 0 (chip idle) to 4 (all CUs are actively running *bench_A*), for both PG enabled and PG disabled cases. Figure 4 shows the measured chip power. We use $P_{idle(CU)}$ and $P_{idle(NB)}$ to denote a CU's idle power and the NB's idle power. For each VF state in the figure, the two bars in the 4CU case have no difference because all CUs are busy and no CU is power gated. In the 3CUs case, the gap between the two bars is $P_{idle(CU)}$. Similarly, in the 2CUs and 1CU cases, the gaps are $2 \times P_{idle(CU)}$ and $3 \times P_{idle(CU)}$, respectively. For the idle case, the difference is $4 \times P_{idle(CU)} + P_{idle(NB)}$ because the NB is also power gated. When the chip is idle and PG is enabled, there is still a constant power, $P_{idle(Base)}$, which exists when the processor is powered on. $P_{idle(Base)}$ does not change with the core VF state. For all VF states, we quantify $P_{idle(CU)}$ and $P_{idle(NB)}$ using this method.

The idle power is partitioned by the busy cores running the application. When PG is enabled, the busy cores in a CU share $P_{idle(CU)}$ and all busy cores in the chip share $P_{idle(NB)} + P_{idle(Base)}$. This is shown in Equation 7, where *m* is the number of the busy cores in a CU and *n* is the number of busy cores in the chip. When PG is disabled, all busy cores share chip idle power, which is always $4 \times P_{idle(CU)} + P_{idle(NB)} + P_{idle(Base)}$. This is shown in Equation 8. Combining this with the per-core dynamic power model, we can derive total per-core power.

$$P_{idle(Core)} = P_{idle(CU)}/m + (P_{idle(NB)} + P_{idle(Base)})/n \quad (7)$$

$$P_{idle(Core)} = (4 \times P_{idle(CU)} + P_{idle(NB)} + P_{idle(Base)})/n \quad (8)$$

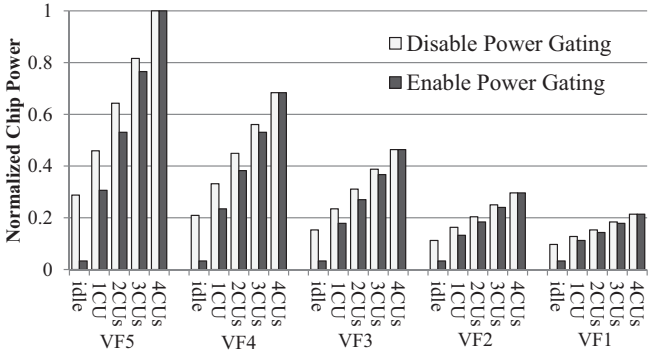


Figure 4 Chip power when power gating is disabled and enabled.

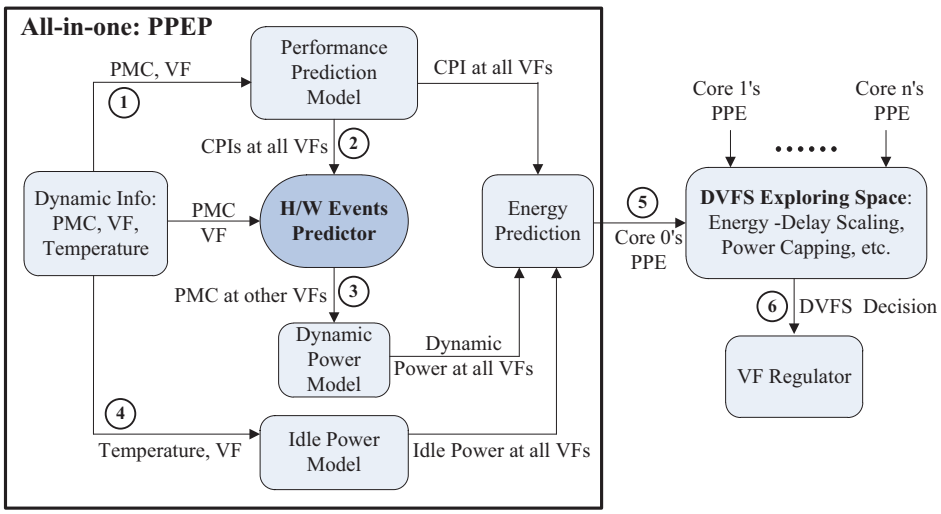


Figure 5 The framework of PPEP and the working flow of PPEP-based DVFS management.

E. Complexity and Generality Analysis

The PPEP framework has four components: a performance prediction model, an idle power model, a dynamic power model, and a power prediction model for estimating power at other VF states. To predict energy consumption, we can simply combine the power prediction model and the performance prediction model.

PPEP can run as a user-level daemon without hardware modifications, and we found that it had negligible overhead at our 200 ms sampling rate. PPEP can also sample faster without significant overhead in the kernel or in firmware. If implemented in firmware, PPEP can also be used to control hardware boost states, which were disabled in our tests.

While we have focused on implementing PPEP on AMD processors, the general techniques should carry between architectures and implementations. CPI predictors have been demonstrated on other architectures [16, 25], for instance. This work could also be taken as a recommendation for the type of performance events that can be useful in other commercial processors.

V. Energy and Power Space Exploration

Figure 5 shows an overview of the PPEP framework. PPEP is a software daemon that runs alongside regular applications and can estimate their performance and power in order to make DVFS decisions. It starts by recording the hardware performance counters, current VF state, and temperature diode values from the cores on the CPU. This is used as the input to the PPEP manager.

From there, PPEP operates as follows: ① it uses the performance predictor to estimate the CPI of the application at all VF states; ② the hardware event predictor, which we described in Section IV-C, takes the predicted CPI values, the current performance counter values, and the VF state and estimates what the hardware events would be at all of the VF states; ③ these event predictions are used to predict the dynamic power that would be consumed at other VF states; ④ the PG-aware idle power model estimates the remaining power at all VF states; ⑤ these values are fed into a decision algorithm; ⑥ this can use these power, energy, and related projections in order to easily make decisions about changing the chip's VF states.

PPEP's predictions can be used to dynamically explore the performance-power-energy (PPE) space to select the optimal VF states within one step. In this section, we demonstrate how PPEP can be used for real-time energy prediction, high responsive (one-step) power capping scheme and further PPE exploration.

A. Energy Prediction

For battery-powered handheld devices, it is important to predict energy consumption in order to make runtime power control decisions that will meet battery life targets. Energy can be calculated by multiplying power and interval length. When using the PPEP framework, we use energy from the current interval to predict energy of the next interval. Besides model fitting errors, phase changes between neighboring intervals can also introduce errors. To examine the total error, we compare the estimated chip energy of the current interval and the measured chip energy of the next interval. We then calculate the AAE across all intervals in each benchmark combination. Figure 6 shows the energy prediction error at VF5 for SPEC® CPU2006 benchmark combinations, with the average AAE (second-to-last bar) on the AMD FX-8320 processor. We also compare our result to the result of the existing work of Green Governors [27] (the last bar) which is also implemented on an AMD processor. The Green Governors power model is based upon a theoretical power model (i.e. CV^2f) and does not consider energy contributions from the NB. As seen in Figure 6, PPEP achieves a 3.6%

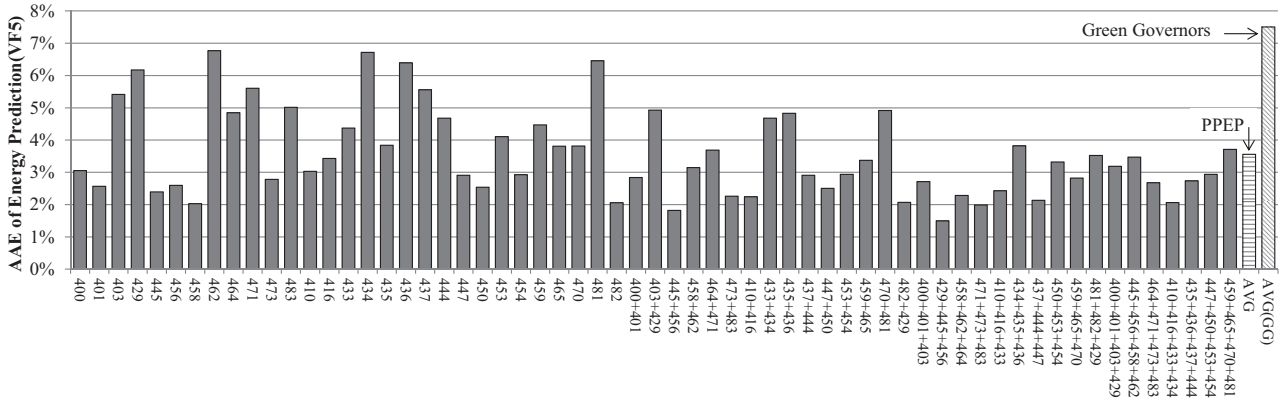


Figure 6 Energy prediction error of the whole chip (including cores and the NB).

average energy prediction error while Green Governors has about 7% average error. We also evaluate the energy prediction error at all other VF states. From VF4 to VF1 the average AAE are 3.3%, 3.7%, 4.0%, and 4.9%, respectively.

B. One-step Power Capping

Power caps are often enforced by temporarily running cores at lower VF states [14, 29, 20, 21]. Finding the VF state that maximizes performance under the power cap is usually an iterative process. A control loop will change the VF state and spend some time determining the current power usage. If the power usage is not yet under the cap, this VF state is lowered and the process repeats. Similarly, if the power is far enough below the cap, the controller will try to raise the VF state in order to gain back performance. In contrast to such an iterative algorithm, PPEP allows proactive power consumption predictions across all VF states. This allows it to directly select the VF state that maximizes performance under the power cap.

Similar to previous work [20, 21], we assume for this experiment that each CU of the processor has a separate power plane, allowing per-CU DVFS. This is in contrast to most current hardware, which supports per-CU frequency scaling but only supports global voltage scaling.

Figure 7 shows the dynamic power capping responsiveness when running a benchmark combination of 429.mcf, 458.sjeng, 416.gamess, and swaptions on four CUs. PPEP adjusts chip power within a single 0.2s sampling period (OS or firmware implementations could respond even faster). In addition, it adheres to the power budget with 94% accuracy. The simple iterative mechanism, which takes 2.8s to adjust to the power budget, adheres to the power budget with 81% accuracy and occasionally violates the power cap. Note that we chose a large power cap swing to demonstrate PPEP's ability to more rapidly reach the cap than iterative algorithms. This is not perfectly representative of all power cap changes, but as an example, removing a laptop from its wall power can cause a significant and rapid power cap change.

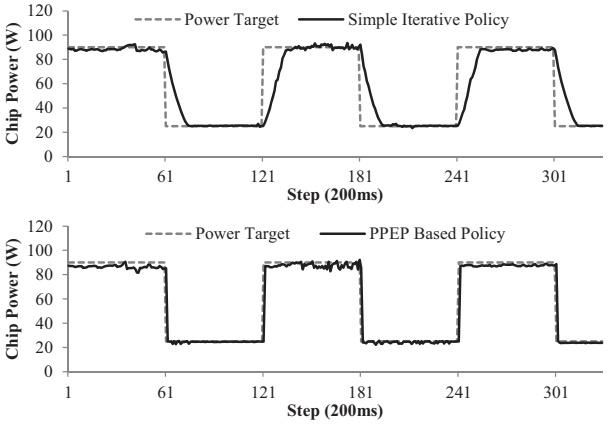


Figure 7 Power capping time responsiveness.

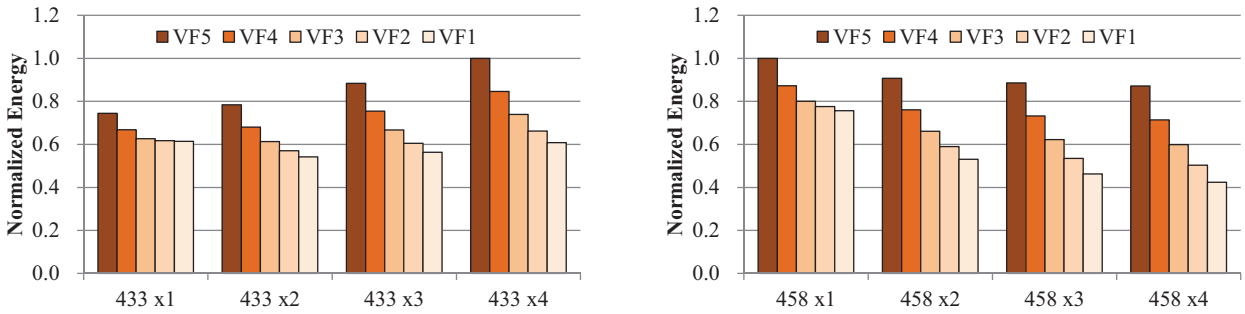
C. Further PPE Exploration

Using the PPEP framework, we are also able to provide insights for future DVFS technology. Note that power gating is enabled for all of these experiments.

1) How background workloads impact energy and EDP: In this experiment, we run the SPEC CPU2006 benchmarks at VF5 with a different number of concurrent instances. We then use PPEP to explore their energy and EDP at other VF states. Of these results, we illustrate 433.milc and 458.sjeng, typical memory- and CPU-bound benchmarks, respectively.

Figure 8 shows the per-thread energy consumption for these programs at different VF states and with a different number of instances. We make three observations:

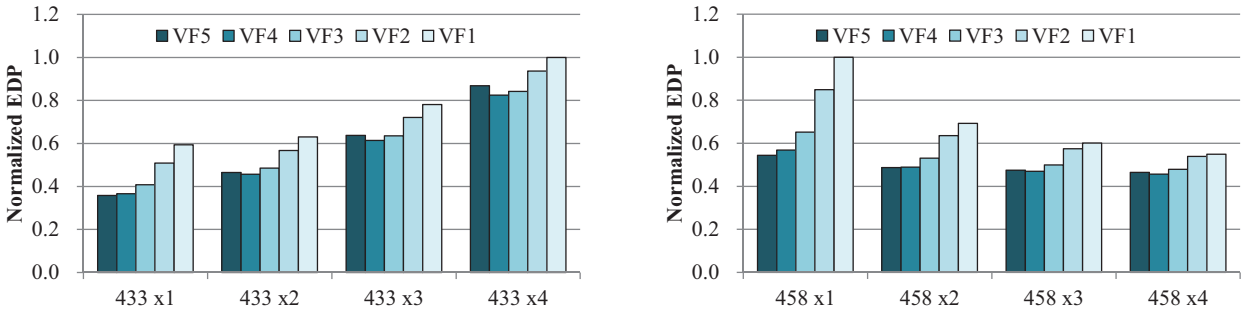
1) For both CPU- and memory-bound applications, running at the lowest VF state produces the lowest energy consumption;
2) At a high VF state, a single memory-bound application consumes less *per-thread* energy than a multi-programmed memory-bound application; and,
3) At the same VF state, a single CPU-bound application consumes more *per-thread* energy than multi-programmed CPU-bound applications.



(a) Per-thread energy of memory-bound 433.milc.

(b) Per-thread energy of CPU-bound 458.sjeng.

Figure 8 Energy comparison (per thread) at different VF states with different background workloads.



(a) Per-thread EDP of memory-bound 433.milc.

(b) Per-thread EDP of CPU-bound 458.sjeng.

Figure 9 EDP comparison (per thread) at different VF states with different background workloads.

The first observation shows that regardless of the number of background workloads, the chip always achieves the least energy at the lowest VF state. Therefore, static VF state policies are useful for energy optimization. Experiments on the AMD FX-8320 and AMD Phenom™ II X6 1090T processors demonstrate that adopting dynamic DVFS policies improves the results by less than 2%.

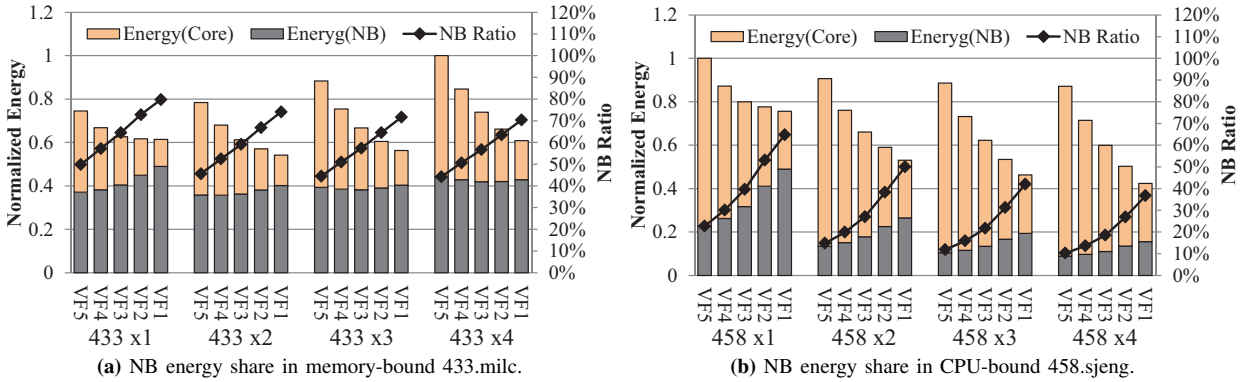
The second observation is caused by shared resource contention in the NB, which leads to longer execution time and higher core static energy. Although there are more threads to share the NB's static power, the increase in execution time caused by memory system contention quickly dissipates any energy savings by requiring more cores to be awake in the high VF state for more time. This is not the case in the low VF states, as they reduce the total energy wasted by core static power.

In the third observation, for CPU-bound applications, there is no contention in the NB. Meanwhile the threads running together share the NB's energy, reducing the per-thread energy across the core.

Figure 9 shows the per-thread EDP for 433.milc and 458.sjeng at different VF states and with a different number of running instances. Memory-bound applications tend to have lower EDP when running alone due to the lack of NB contention. CPU-bound applications tend to have lower EDP when running with more instances from similar applications because they can share chip-wide static power. Another observation is that, with the increasing number of background threads, the VF state for best EDP shifts from VF5 to VF4.

In summary, background workloads have significant impact on an application's energy and EDP. Dynamic DVFS policies should also carefully consider background workloads and adjust VF states for the best energy and/or EDP.

2) How does the NB impact energy savings: Recent work on simulated big.LITTLE architecture for mobile workloads [11] underscored that the uncore part of the chip also plays an important role and highlighted the need for uncore scalability to improve energy efficiency. Since PPEP is able to provide separate core energy and NB energy estimates, we use them to explore the potential benefit of multiprocessors to have an NB with multiple VF states. Figure 11 shows the per-thread energy of 433.milc and 458.sjeng and breaks it down by core and NB energy. For memory-bound applications, the NB consumes 60% of the total energy on average, and a minimum of 45% of the total energy. For CPU bound applications, the NB consumes much less energy—25% of the total on average, and a minimum of 10%. When the number of busy CUs is small, the NB consumes a higher percentage of total energy because there are fewer busy cores to share the NB energy consumption. When running at a lower VF state, the NB's fraction increases. Lowering a core's voltage does not reduce the dynamic energy used by the NB, but it increase the program's execution time, which does increases the NB's energy usage.



(a) NB energy share in memory-bound 433.milc. **(b)** NB energy share in CPU-bound 458.sjeng.

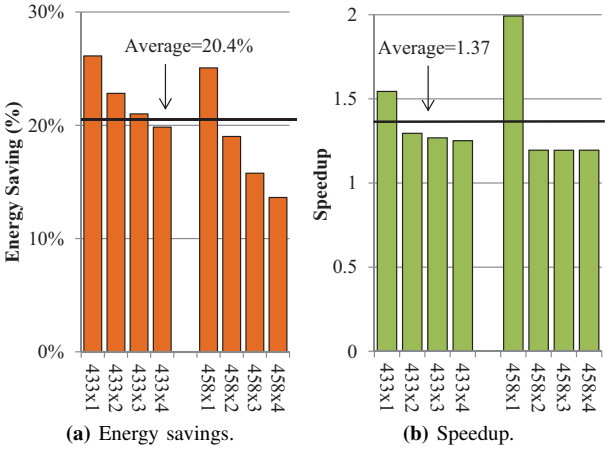
Figure 10 NB energy consumption percentage.

This shows that it is important to evaluate the energy benefit of scaling the NB voltage and frequency. As such, we use PPEP to conduct studies assuming two VF states for the NB. The higher one is the current VF_hi (1.175V, 2.2GHz). We introduce a lower VF state to the NB with a 20% voltage drop and a 50% frequency drop as VF_lo (0.940V, 1.1GHz). We assume the idle power of the NB drops 40%, and dynamic power for same operations drops 36%. For performance prediction, we assume the leading loads cycles increase by 50% when frequency drops from high to low. We adopt these considerations and re-evaluate the PPE of the new Core-NB VF combinations.

As can be seen in Figure 11(a), both memory- and CPU-bound applications can achieve further energy saving through scaling the VF state of the NB. We compare the scaling range of energy consumption when fixing the run mode to x1, x2, x3 or x4. For 433.milc, the potential of energy savings with NB VF scaling are 26%, 23%, 21%, and 20%, respectively, with NB VF scaling. For 458.sjeng these values are 25%, 19%, 16%, and 14%. For memory-bound applications, a lower voltage makes a large reduction in the NB's dynamic energy. Although the increase in execution time causes a larger portion of the program's energy to come from the NB idle energy, it still reduces chip-wide energy usage. The energy benefits are also significant for CPU-bound applications, which may run slightly slower but see much less static NB energy usage.

Figure 11(b) shows that scaling the NB can reduce execution time with similar energy consumption. We observe that, because the NB uses less power, the cores can now run at a higher VF state. This reduces their execution time without changing chip-wide energy usage. Using core-VF1 and NB-VF_hi as the baseline, we observe that for the memory bound 433.milc, the potential performance gains are 1.54×, 1.30×, 1.27×, and 1.25×. For the CPU-bound 458.sjeng, these values are 1.99×, 1.19×, 1.19×, and 1.20×.

Note that due to measurement limitation, we cannot separate the power of the memory controller from that of the last-level cache in our NB. As such, we scale both in the above results. In reality, for SRAM reliability reasons, only the memory controller power should be scaled.



(a) Energy savings. **(b)** Speedup.

Figure 11 Further energy savings with a scalable NB and the performance speedup with similar energy consumption.

This experiment indicates that future DVFS technology should also consider scaling the NB to achieve energy savings and higher performance.

VI. Related Work

Research on power modeling has evolved from simple frequency and voltage scaling to hardware event counter-based regression models. Many other empirical models require off-line profiling or preprocessing for each application, which do not provide flexibility and generality for new workloads. There are many studies that use linear power models built from different performance counters combinations. Karan et. al. [26] propose a real-time power model based on four hardware event counters on AMD Phenom™ quad-core processor. Isci et al. [15] collected hardware event information to create a fingerprint of the power traces, but their particular microprocessor required many counters, and was thus quite costly. Deng et al. [6] proposed a simulation based system-level framework, CoScale, with performance and power predictions. The recent work on Green Governors

built a power model from a theoretical (CV^2f) power formula by calculating the effective capacitance and the processor's dynamic activity [27]. This work measured the voltage regulator to get the power traces for cores but did not consider the NB for DVFS benefits. In contrast, PPEP builds performance and power models based on hardware measurements and extends them to predict across VF states by implementing a hardware event predictor. Once the model is built for a specific processor, runtime usage does not require power meters or off-line application training. It simply follows the application's behavior with high sensitivity.

Power capping has been achieved by migrating threads [5] or scaling down core frequency [20, 21, 29] to meet the power budget. Steepest Drop [29] assumes knowledge of the power consumption of each core, which is not yet fully supported by modern processors. Others [20, 21] have developed the three-layer power control architecture called FreqPar for many-core processors, which requires a power monitor attached to the chip. Pack & Cap [5] predicts performance with a multi-nomial logistic regression (MLR) classifier trained through an offline analysis. When queried at runtime, this classifier returns the best candidate operating point. In contrast, PPEP predicts performance and power directly from architectural properties, providing more general and theoretical insights with a low complexity. This ability, especially for distant VF states, makes PPEP a good infrastructure to implement single step power capping policies in commercial processors without attaching external power monitors.

VII. Conclusion

In this paper, we propose the PPEP framework to provide online PPE estimations across all available voltage-frequency states. PPEP implements a hardware event predictor that takes the performance counters from one VF state and uses them to generate dynamic information about another VF state. We verify PPEP's modeling accuracy on both an AMD FX-8320 and an AMD Phenom™ II X6 1090T processor with a broad set of benchmarks, including SPEC® CPU2006, PARSEC, and the NAS Parallel Benchmarks.

Using PPEP, we conduct an energy and power space exploration which demonstrates the following: PPEP is able to make power adjustment decisions that meet the target power cap in a single step, improving the response time compared to a commonly-practiced iterative power capping mechanism. We also show that background workloads have significant impact on application's energy and EDP. Thus it is an important factor to consider in DVFS policies. Finally, we show that future processor designs which take advantage of scalable VF states in the north bridge, can offer an extra 20% energy saving or 1.4× performance improvement versus modern cores which maintain mostly constant north bridge VF states.

Acknowledgment

We would like to thank our shepherd and anonymous reviewers for their comments and suggestions that have greatly improved this paper. In addition, we wish to thank Y. Eckert and I. Paul for their help and advice. Finally, thanks to V. Spiliopoulos for his help in understanding the Green Governors model. This work is partially supported by 863 Program of China (2012AA010905), NSFC (61472431). Bo Su is supported by NUDT/Hunan Innov. Fund. For PostGrad. (B120604, CX2012B029).

References

[1] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Finebert, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS Parallel Benchmarks," Tech. Rep. RNR-94-007, March 1994.

[2] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second ed.* Morgan & Claypool, 2013.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proc. of the Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[4] T. Cao, S. M. Blackburn, T. Gao, and K. S. McKinley, "The Yin and Yang of Power and Performance for Asymmetric Hardware and Managed Software," in *Proc. of the Int'l Symp. on Computer Architecture (ISCA)*, 2012.

[5] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps," in *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2011.

[6] Q. Deng, D. Meisner, A. Bhattacharjee, T. Wenisch, and R. Bianchini, "CoScale: Coordinating CPU and Memory System DVFS in Server Systems," in *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2012.

[7] H. Esmaeilzadeh, T. Cao, Y. Xi, S. M. Blackburn, and K. S. McKinley, "Looking Back on the Language and Hardware Revolutions: Measured Power, Performance, and Scaling," in *Proc. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[8] S. Eyerman and L. Eeckhout, "A Counter Architecture for Online DVFS Profitability Estimation," *IEEE Trans. on Computers*, vol. 59, no. 11, pp. 1576–1583, 2010.

[9] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A Mechanistic Performance Model for Superscalar Out-of-Order Processors," *ACM Transactions on Computer Systems*, vol. 27, no. 2, pp. 3:1–3:37, 2009.

[10] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[11] V. Gupta, P. Brett, D. Koufaty, D. Reddy, S. Hahn, K. Schwan, and G. Srinivasa, "The Forgotten 'Uncore': On the Energy-

Efficiency of Heterogeneous Cores," in *Proc. of the USENIX Annual Technical Conf. (USENIX ATC)*, 2012.

[12] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[13] W. Huang, C. Lefurgy, W. Kuk, A. Buyuktosunoglu, M. Floyd, K. Rajamani, M. Allen-Ware, and B. Brock, "Accurate Fine-Grained Processor Power Proxies," in *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2012.

[14] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2006.

[15] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," in *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2003.

[16] R. Joseph and M. Martonosi, "Run-time Power Estimation in High Performance Microprocessors," in *Proc. of the Int'l Symp. on Low Power Electronics and Design (ISLPED)*, 2001.

[17] G. Keramidas, V. Spiliopoulos, and S. Kaxiras, "Interval-Based Models for Run-Time DVFS Orchestration in Superscalar Processors," in *Int'l Conf. on Computing Frontiers (CF)*, 2010.

[18] C. Lefurgy, X. Wang, and M. Ware, "Power Capping: a Prelude to Power Shifting," *Cluster Computing*, vol. 11, no. 2, pp. 183–195, 2008.

[19] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy," in *Proc. of the Conf. on Design, Automation and Test in Europe (DATE)*, 2007.

[20] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable Power Control for Many-Core Architectures Running Multi-threaded Applications," in *Proc. of the Int'l Symp. on Computer Architecture (ISCA)*, 2011.

[21] K. Ma and X. Wang, "PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs," in *Proc. of the Int'l. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2012.

[22] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, "Predicting Performance Impact of DVFS for Realistic Memory Systems," in *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2012.

[23] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili, "Cooperative Boosting: Needy versus Greedy Power Management," in *Proc. of the Int'l Symp. on Computer Architecture (ISCA)*, 2013.

[24] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mikherjee, B. R. Sheikh, and S. Yardi, "CAMP: A Technique to Estimate Per-Structure Power at Run-time using a Few Sample Parameters," in *Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2009.

[25] B. Rountree, D. K. Lowenthal, M. Schulz, and B. R. de Supinski, "Practical Performance Prediction Under Dynamic Voltage Frequency Scaling," in *Int'l Green Computing Conf. and Workshops (IGCC)*, 2011.

[26] K. Singh, M. Bhadauria, and S. A. McKee, "Real Time Power Estimation and Thread Scheduling via Performance Counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[27] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green Governors: A Framework for Continuously Adaptive DVFS," in *Proc. of the Int'l Green Computing Conference and Workshops (IGCC)*, 2011.

[28] B. Su, J. L. Greathouse, J. Gu, M. Boyer, L. Shen, and Z. Wang, "Implementing a Leading Loads Performance Predictor on Commodity Processors," in *Proc. USENIX Annual Technical Conf. (USENIX ATC)*, 2014.

[29] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable Thread Scheduling and Global Power Management for Heterogeneous Many-Core Architectures," in *Proc. Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2010.